How exactly do you compute the Fast Fourier Transform?

Asked 13 years, 4 months ago Modified 7 years, 1 month ago Viewed 23k times

16

I've been reading a lot about Fast Fourier Transform and am trying to understand the low-level aspect of it. Unfortunately, Google and Wikipedia are not helping much at all.. and I have like 5 different algorithm books open that aren't helping much either.

I'm trying to find the FFT of something simple like a vector [1,0,0,0]. Sure I could just plug it into Matlab but that won't help me understand what's going on underneath. Also, when I say I want to find the FFT of a vector, is that the same as saying I want to find the DFT of a vector just with a more efficient algorithm?

algorithm math fft

Share Improve this question Follow

edited Jul 12, 2010 at 20:06
ShreevatsaR
38.5k 17 105 126

asked Jul 11, 2010 at 22:17
Parth
245 2 4 7

While it may not help in understanding, here is a good implementation: fftw.org. It is fairly well-documented. – Adam Shiemke Jul 12, 2010 at 14:42

Note that this is a two-part question: 1. How can I intuitively imagine or even predict what the Discrete Fourier Transform (as mentioned in the answers, FFT is really an algorithm for performing the DFT) does to my input? and 2. How do I implement FFT? – Domi Dec 2, 2013 at 4:50

So what is it for? The Fourier Transform transforms an input signal into frequency space, which tells you how often different frequencies appear in your signal. This gives you a lot of information about your signal that you can use to find, eliminate or amplify certain frequencies and even other properties of your signal. This is possible because the Fourier Transform has an inverse that allows you to convert your changed frequency space back to signal space. – Domi Dec 2, 2013 at 4:57

5 Answers

Sorted by: Highest score (default)

28

You're right, "the" Fast Fourier transform is just a name for *any* algorithm that computes the discrete Fourier transform in O(n log n) time, and there are several such algorithms.

Here's the simplest explanation of the DFT and FFT as I think of them, and also examples for small N, which may help. (Note that there are alternative interpretations, and other algorithms.)

Discrete Fourier transform

Given `N` numbers f_0, f_1, f_2, ..., f_{N-1}, the DFT gives a different set of `N` numbers.

Specifically: Let ω be a primitive *N*th root of 1 (either in the complex numbers or in some finite field), which means that $\omega^N=1$ but no smaller power is 1. You can think of the f_k's as the coefficients of a polynomial $P(x) = \sum f_k x^k$. The *N* new numbers F_0, F_1, ..., F_{N-1} that the DFT gives are the results of **evaluating the polynomial** at powers of ω. That is, for each *n* from 0 to N-1, the new number F_n is $P(\omega^n) = \sum_{0\le k\le N-1} f_k\omega^{nk}$.

$$F_n = P(\omega^n) = \sum_{k=0}^{N-1} f_k \omega^{nk}$$

[The reason for choosing ω is that the inverse DFT has a nice form, very similar to the DFT itself.]

Note that finding these F's naively takes $O(N^2)$ operations. But we can exploit the special structure that comes from the ω's we chose, and that allows us to do it in O(N log N). Any such algorithm is called the fast Fourier transform.

Fast Fourier Transform

So here's one way of doing the FFT. I'll replace N with 2N to simplify notation. We have f_0, f_1, f_2, ..., f_{2N-1}, and we want to compute $P(\omega^0)$, $P(\omega^1)$, ... $P(\omega^{2N-1})$ where we can write

$P(x) = Q(x) + \omega^N R(x)$ with

$Q(x) = f_0 + f_1x + ... + f_{N-1}x^{N-1}$

$R(x) = f_N + f_{N+1}x + ... + f_{2N-1}x^{2N-1}$

Now here's the beauty of the thing. Observe that the value at ω^{k+N} is very simply related to the value at ω^k:
$P(\omega^{k+N}) = \omega^N(Q(\omega^k) + \omega^N R(\omega^k)) = R(\omega^k) + \omega^N Q(\omega^k)$. So the evaluations of Q and R at ω^0 to ω^{N-1} are enough.

This means that your original problem — of evaluating the 2N-term polynomial P at 2N points ω^0 to ω^{2N-1} — has been reduced to the two problems of evaluating the N-term polynomials Q and R at the N points ω^0 to ω^{N-1}. So the running time $T(2N) = 2T(N) + O(N)$ and all that, which gives $T(N) = O(N \log N)$.

Examples of DFT

Note that other definitions put factors of 1/N or $1/\sqrt{N}$.

For N=2, ω=-1, and the Fourier transform of (a,b) is (a+b, a-b).

For N=3, ω is the complex cube root of 1, and the Fourier transform of (a,b,c) is (a+b+c, $a+b\omega+c\omega^2$, $a+b\omega^2+c\omega$). (Since $\omega^4=\omega$.)

For N=4 and ω=i, and the Fourier transform of (a,b,c,d) is (a+b+c+d, a+bi-c-di, a-b+c-d, a-bi-c+di). In particular, the example in your question: the DFT on (1,0,0,0) gives (1,1,1,1), not very illuminating perhaps.

Share Improve this answer Follow

edited Feb 8, 2017 at 14:28
Community Bot
1 1

answered Jul 12, 2010 at 7:50
ShreevatsaR
38.5k 17 105 126

So much clarity in this answer +1 – jmishra Mar 9, 2013 at 8:18

7

The FFT is just an efficient implementation of the DFT. The results should be identical for both, but in general the FFT will be much faster. Make sure you understand how the DFT works first, since it is much simpler and much easier to grasp.

When you understand the DFT then move on to the FFT. Note that although the general principle is the same, there are many different implementations and variations of the FFT, e.g. decimation-in-time v decimation-in-frequency, radix 2 v other radices and mixed radix, complex-to-complex v real-to-complex, etc.

A good practical book to read on the subject is Fast Fourier Transform and Its Applications by E. Brigham.

Share Improve this answer Follow

answered Jul 11, 2010 at 22:23
Paul R
210k 37 393 563

1 +1 for the Brigham reference. It's the best explanation I've ever read. – andand Jul 13, 2010 at 21:48

@andand: thanks, yes, excellent book, even though it's quite old now: the applications chapters are good too, and still relevant. – Paul R Jul 13, 2010 at 21:52

2

Yes, the FFT is merely an efficient DFT algorithm. Understanding the FFT itself might take some time unless you've already studied complex numbers and the continuous Fourier transform; but it is basically a base change to a base derived from periodic functions.

(If you want to learn more about Fourier analysis, I recommend the book *Fourier Analysis and Its Applications* by Gerald B. Folland)

Share Improve this answer Follow

edited Jul 11, 2010 at 22:30

answered Jul 11, 2010 at 22:22

You
23k 3 51 64

2

I'm also new to Fourier transforms and I found this online book very helpful:

The Scientists and Engineer's Guide to Digital Signal Processing

The link takes you to the chapter on the Discrete Fourier Transform. This chapter explains the difference between all the Fourier transforms, as well as where you'd use which one and pseudocode that shows how you go about calculating the Discrete Fourier Transform.

Share Improve this answer Follow

answered Aug 4, 2010 at 9:33

Garg Unzola
376 1 4 9

2

If you seek a plain English explanation of DFT and a bit of FFT as well, instead of academic *goggledeegoo*, then you **must** read this: http://blogs.zynaptiq.com/bernsee/dft-a-pied/

I couldn't have explained it better myself.

Share Improve this answer Follow

edited Oct 11, 2016 at 17:00

Uli Köhler
13.1k 16 70 122

answered Sep 28, 2010 at 21:29

Rob Vermeulen
1,940 1 15 22

Shame there's no explanation of the actual FFT algorithm. Great link anyway, thanks! – macbirdie Nov 7, 2010 at 21:38